Pricing Online Decisions: Beyond Auctions

Alon Ardenboim[1], Ilan Reuven Cohen[1], Łukasz Jeż[1,2] and Amos Fiat[1]

[1]Blavatnik School of Computer Science, Tel-Aviv University*
[2]University of Wrocław, Institute of Computer Science†
{*alonarden,ilanrcohen*}*@gmail.com, lje@cs.uni.wroc.pl, fiat@tau.ac.il*

July 7, 2014

"You pays your money and you takes your choice."
— Mark Twain, *Huckleberry Finn*

Abstract

We consider dynamic pricing schemes in online settings where selfish agents generate online events. Previous work on online mechanisms has dealt almost entirely in auction settings with the goal of maximizing social welfare or revenue. This paper deals with quite general settings and minimizing social costs. We show that appropriately computed posted prices allow one to achieve essentially the same performance as the best online algorithm. This holds in a wide variety of settings. Unlike online algorithms that learn about the event, and then make enforcable decisions, prices are posted without knowing future events and are inherently dominant strategy incentive compatible.

In particular we show that one can give efficient posted price mechanisms for metrical task systems, some instances of the k-server problem, and metrical matching problems. We give both deterministic and randomized algorithms. Such posted price mechanisms decrease the social cost dramatically over selfish behavior where no decision incurs a charge. One alluring application of this is reducing the social cost of free parking exponentially.

*Partially supported by the Israeli Centers of Research Excellence (I-CORE) program, (Center No.4/11) and the Google Inter-University Center.

†Partially supported by the Foundation for Polish Science (FNP) START Scholarship and the Polish National Science Center (NCN) Grant DEC-2013/09/B/ST6/01538.

1 Introduction

In many scenarios, agents arrive and make decisions which influence the global environment. Such decisions affect the welfare and options of subsequent agents and may be detrimental to the social welfare if made selfishly. Examples abound, pollution of the commons, aggressive driving and aftermaths, using public transportation vs driving ones own car, choosing a driving route, etc.

We consider a variety of online settings and devise dynamic posted pricing schemes for reducing the social costs, to (nearly) the minimal social costs possible. Posted prices (or rewards) are easily understood by consumers and are in wide use: item prices in a supermarket, parking fees, toll roads, grants for socially beneficial behavior, etc.

Dynamic posted prices are also in wide use, of particular interest are varying tolls on express lanes and the SFPark system (SFpark.org) which sets parking prices in San Francisco as a function of parking congestion [14, 6]. Interestingly, our dynamic pricing scheme can be viewed as a refinement of the SFPark system. One could take the SFPark pricing (which sets fixed pricing per city block) and add modulation based upon the parking pattern along that block. This would improve social welfare by an exponential factor (in expectation, in theory).

In particular we consider dynamic posted pricing schemes to reduce the social costs in

1. Selfish Metrical Task systems, where tasks are associated with selfish agents who seek to minimize the sum of the state transition costs, the work associated with the task in the state chosen, and the (posted) surcharge for choosing a specific state. We give a dynamic posted pricing scheme that is $O(m)$ competitive.
2. Selfish k-server requests, where the request is issued by a selfish agent, where agents choose to minimize the sum of the distance traversed by the server plus the (posted) surcharge for choosing a specific server. We give a dynamic posted pricing scheme that is $O(1)$ competitive for two servers in arbitrary metric spaces and k competitive for k servers on a line. We remark that the k-server problem on finite metric spaces can be expressed as a metrical task system but the resulting competitive ratio for task systems is much worse that $O(k)$.
3. Selfish parking, where agents seek to minimize the sum of the distance to the destination plus the (posted) parking fee. We give a dynamic posted pricing scheme that is $O(\log(R))$ competitive where R is the ratio between the largest and smallest pairwise distance amongst parking spaces. This is an exponential improvement over the competitive ratio of free parking. Our parking problem cannot be expressed as a metrical task system[1].

The following are related (but distinct) concepts:

1. Online algorithms where events appear and are dealt with in a centralized fashion. It is assumed that events are "true". Online algorithms enforce their decisions. There is no sense in which an event says "I don't like what you've decided in my case."
2. Online mechanisms where events appear and decisions are made, prizes and fees set, in a centralized fashion. Just as with competitive algorithms, online mechanisms can (magically?) enforce their decisions. There is no sense in which an event can "revolt" against a decision that she does not like.
3. Dynamic pricing (this paper) sets a posted price for every possible decision. Incoming selfish events make their own decisions based upon their own self interest. "You pays your money and you makes your choice".

In contrast to online mechanisms (or online algorithms) that inherently expect the agent (event) to reveal its type (arrive) in it's entirety, dynamic posted pricing schemes ask nothing of the agent but simply observe

[1] Perhaps if one allowed cars to be towed away we could express the problem as an asymmetric task system.

what the agent actually does. For task systems, the pricing scheme observes the task cost *in the state actually chosen by the agent*, for parking, the pricing scheme observes *where she actually parked.*

Most of the work done in online mechanism design is in the context of auctions and specifically in maximizing revenue or social welfare. The settings we consider, dynamic posted prices for task systems, k-servers and pricing, are not in this context. We are reducing social cost, not maximizing social welfare.

2 Related Work

Lavi and Nisan [11] initiated the study of online auctions. The consider a model of indivisible goods where bidders have a privately known valuation for different quantities of the goods. Agents learn their valuation at some time and must bid immediately. The auction must decide immediately how many units to allocate and the price. They show that the only dominant strategy incentive compatible mechanisms are those that offer a price menu for different quantities.

Awerbuch, Azar, and Myerson [1] give a general scheme that produces posted prices for general combinatorial auctions, with a competitive ratio equal to the logarithm of the ratio between highest and lowest prices, times the underlying competitive ratio for the combinatorial auction (which is terrible, in general).

There has been a large body of work on online auctions in the setting where the order of events is a uniformly random permutation. Hajiaghayi, Kleinberg and Parkes [8] discuss limited supply online auctions, and Babaioff, Immorlica, Kempe, and Kleinberg [2] that introduced the knapsack and matroid secretary problems, about which there has been much subsequent work.

Parkes [13] discusses a general model for dynamic environments and online mechanism design. In this general model a mechanism makes (and enforces) a sequence of decisions. The decisions may depend on interaction with agents as well as changes to the environment. Unfortunately, the positive results in this general setting have been limited to so-called "single valued preferences" where an agent gets a fixed value r iff one of a set of "interesting decisions" in made between her arrival and departure. None of the problems we consider are single valued.

3 Pricing States for Metrical Task Systems

A metrical task system [3] is described by a symmetric non-negative $m \times m$ matrix, $(d_{st})_{1 \le s,t \le m}$, where $d_{ss} = 0$ and the triangle inequality holds: $d_{st} \le d_{sk} + d_{kt}$ for all $1 \le s,t,k \le m$. The set of states $S = \{1,\ldots,m\}$.

The selfish metrical task system is a variant of the metrical task system where tasks are associated with agents. Agents arrive online, the ith agent to arrive has some task to perform, represented by a vector of m non negative real values, $w^i = \left(w^i_1, w^i_2, \ldots, w^i_m\right)$. The task w^i is private to agent i.

The initial state of the system is S_0, the state of the system may change over time. Let S_{i-1} be the state of the system upon the arrival of agent i. The ith agent to arrive may decide to change the state of the system upon her arrival to some other state (*i.e.*, $S_i \neq S_{i-1}$) or not (*i.e.* $S_i = S_{i-1}$).

Given states s, t, and a task $w = (w_1, w_2, \ldots, w_m)$, define $C(s,t,w) = w_t + d_{st}$, *i.e.* the cost to switch states from s to t and process task w in state t.

A dynamic pricing scheme for selfish metrical task systems sets prices (surcharges) on the states of the system. Formally, a dynamic pricing scheme $P = (P_1, P_2, \ldots)$ is a sequence of price functions $P_0, P_1, \ldots$, where the ith function

$$P_i : \left\langle \left(S_1, w^1_{S_1}\right), \left(S_2, w^2_{S_2}\right), \ldots, \left(S_i, w^i_{S_i}\right) \right\rangle \times S \mapsto \mathbb{R}^+.$$

The information available to price function P_i is the choices made by agents $1 \le j \le i$, (agent $1 \le j \le i$ choose state S_j), and the work done by agent $1 \le j \le i$ in state S_j ($w^j_{S_j}$). This is represented above by the

sequence $\left\langle \left(S_1, w^1_{S_1}\right), \left(S_2, w^2_{S_2}\right), \ldots, \left(S_i, w^i_{S_i}\right)\right\rangle$. The prices computed by pricing function P_i are relevant for agent $i+1$.

For notational convenience, we use P_i as a pricing function over the set of states after i tasks are processed, where the "history" of the system is implicit. We stress that the pricing is set without any information regarding future events.

Note that pricing function P_i *cannot* depend on tasks w^j, $j > i$, (this is distinct from an online algorithm that may depend on w^{i+1}). Moreover, it knows very little about $w^1, w^2, \ldots, w^i$. It only knows the *decisions* made by the agents and the actual work done, *i.e.*, S_j and $w^j_{S_j}$ for $1 \leq j \leq i$. In particular, this means that the pricing scheme does not know the optimal cost to process tasks $w^1, \ldots, w^i$ and cannot compute the work function for a state.

The goal of a selfish agent is to to minimize her disutility. This is the sum of the following components:

1. The work required to do the task in the state selected by the agent. For agent i this is $w^i_{S_i}$.
2. The transition cost to change states. For agent i this is $d_{S_{i-1}S_i}$.
3. The surcharge associated with choosing (or remaining in) some state, this is determined by the pricing scheme (see below). For agent i this is $P_{i-1}(S_i)$.

Let P be some dynamic pricing scheme, $m \in Z^+$, and $M = (d_{ij})_{1 \leq i,j \leq m}$ a metrical task system. Because these are posted prices, and because agents may not change their position in the seqeuence, truthfulness is not an issue. Given any sequence of agent tasks $w^1, w^2, \ldots$, it is a dominant strategy for agents to minimize their disutility. Let $\mathbf{w} = w_1, \ldots, w_n$ be a finite sequence of tasks. The sequence of states in the dominant strategy equilibria, $S_0, S_1, S_2, \ldots, S_n$ is not entirely defined because of possible ties (two different decisions give the same minimal disutility).

Let $S_{\text{eq}}(P, M, w)$ be the set of all sequences in equilibria for the pricing scheme P, the metrical task system M, and the task sequence w. Also, define $S^*(M,w) = S^*_0(M,w), S^*_1(M,w), S^*_2(M,w), \ldots, S^*_n(M,w)$ (where $S^*_0(M,w) = S_0$ — the initial state) to be the sequence of states that minimize the total cost

$$\sum_{i=1}^{n} C(S^*_{i-1}(M,w), S^*_i(M,w), w^i).$$

For a metrical task system M, and a finite sequence of tasks $\mathbf{w} = w^1, \ldots, w^n$, define

$$\text{OPT}(M, \mathbf{w}) = \sum_{i=1}^{n} C(S^*_{i-1}(M,w), S^*_i(M,w), w^i).$$

We define the price of anarchy of a pricing scheme P with respect to a class of metrical task systems Ψ, $PoA(P, \Psi)$ to be

$$PoA(P,\Psi) \quad = \quad \sup_{M \in \Psi, \mathbf{w}, R \in S_{\text{eq}}(P,M,\mathbf{w})} \frac{\sum_{i=1}^{n} C(R_{i-1}, R_i, w^i) - C(M)}{\text{OPT}(M, \mathbf{w})}, \tag{1}$$

where $C(M)$ above is some constant that may depend on the transition cost matrix M. The price of anarchy of a pricing scheme P without restricting M to any class of MTS, $PoA(P)$, is defined analogously.

3.1 The Fractional Traversal Algorithm for Metrical Task Systems

We now describe the fractional traversal algorithm for metrical task systems from [3]. The fractional traversal algorithm is an imaginary algorithm (it does not follow the rules of metrical task systems) in that it may perform a task by performing fractions thereof in may different states, and traversing between states multiple times. Moreover, it may even return to a state an exit the same state multiple times during the execution of a single task.

We use the following terms:

- A traversal sequence, $\tau = \tau_1, \tau_2, \ldots$, is an infinite sequence of states of the task system, where every state of the task system may appear several times in the sequence. Note that this is any sequence and not necessarily the traversal sequence produced in [3].
- Given such a traversal sequence $\tau = \tau_1, \tau_2, \ldots$, define the traversal distance between indices ℓ and ℓ' to be $\delta_{\ell,\ell'} = \delta_{\ell',\ell} = \sum_{j=\min(\ell,\ell')}^{\max(\ell,\ell')-1} d_{\tau_j,\tau_{j+1}}$.

In the full version, we describe the [3] fractional traversal algorithm on some traversal sequence τ. The fractional traversal algorithm computes a sequence of indices $t_0, t_1, \ldots$, where t_i is the position of the fractional traversal algorithm index within the traversal sequence after processing task i. $t_0 = 1$ initially.

We now show that a traversal algorithm's performance is monotone with respects to the tasks it receives.

Lemma 1. *Let τ be some traversal sequence, and let $\mathbf{w} = \langle w^1, \ldots, w^n \rangle$ and $\tilde{\mathbf{w}} = \langle \tilde{w}^1, \ldots \tilde{w}^n \rangle$ be two sequences of tasks such that for every $i \in \{1, \ldots, n\}$ and every $s = 1, \ldots, m$, we have that $\tilde{w}^i_s \leq w^i_s$. The total work done by the traversal algorithm with respect to τ and $\mathbf{w}$ is at least the amount of work done with respect to τ and $\tilde{\mathbf{w}}$.*

As an immediate corollary we get from this lemma is that the total cost of the fractional traversal algorithm given task sequence $\tilde{\mathbf{w}}$ is smaller than the cost of the fractional traversal algorithm given task sequence $\mathbf{w}$. This is since it $\delta_{\tilde{t}_0,\tilde{t}_i} \leq \delta_{t_0,t_i}$ for every $i \in \{0, \ldots, n\}$.

Borodin et al. showed the following with regard to the traversal algorithm:

Theorem 2 (Borodin et al. [3])**.** *For metrical task system there exists a traversal sequence τ for which the fractional traversal algorithm achieves a competitive ratio of $8(m-1)$.*

3.2 "Follow the Traversal" Algorithm

We would like to come up with a dynamic pricing scheme for metrical task systems for which selfish agents, driven by self interest, follow the traversal sequence.

We now describe an new algorithm which can be simulated by a dynamic pricing scheme that encourages greedy agents to "follow" the fractional traversal algorithm. The algorithm is a given metrical task system (a set of states S, and a distance metric $d : S \times S \mapsto \mathbb{R}^+$), a traversal sequence τ, and processes an online sequence of tasks $w^1, w^2, \ldots$.

Roughly speaking, after processing each task, the algorithm simulates the traversal algorithm presented in Section 3.1. When handling task $i+1$, the algorithm views all states between its current position in the sequence, and the fractional traversal algorithm's current position in the sequence, t_i, as begin one super state where the cost of a task in a super state is the min cost in any constitute state, cf. Algorithm 1

Even though there may be several indices that minimize $\tilde{c}$, tie breaking can be done arbitrarily, since all the claims hold for every index which minimizes $\tilde{c}$.

Theorem 3. *For any traversal sequence τ, distance metric d, and sequence of tasks $w^1, \ldots, w^n$, let $C^{A2} = w^{A2} + d^{A2}$ be the total cost of Algorithm 1, and $C^T = w^T + d^T$ the total cost of the traversal algorithm for that sequence of tasks (the processing cost plus the transition cost). It must be that $C^{A2} \leq 2 \cdot C^T$.*

3.3 Pricing Scheme

In this section, we define a dynamic pricing scheme that observes decisions made by greedy agents and sets a surcharge for the next agent to arrive. We want to set prices so that states chosen by (greedy) agents are the same as the ones chosen by Algorithm 1 (up to tie breaking) given the sequence of tasks $\mathbf{w}$.

The problem is that dynamic pricing schemes, as we've defined them, can only observe the greedy behavior of agents — the state chosen by the agents, and the amount of work done in that state while processing

Input: A traversal sequence τ, a set of states S, a distance metric d and an online sequence of tasks $w^1, w^2, \ldots$.
Let ℓ_i denote the position of the algorithm in the traversal sequence after completing tasks $w^1, \ldots, w^i$ and t_i be the position of the traversal algorithm after completing these tasks ($\ell_0 = t_0 = 1$).
Given task w^{i+1}:

1. Let $j_{\min} \leftarrow \min(\ell_i, t_i)$ and $j_{\max} \leftarrow \max(\ell_i, t_i)$.
2. Let $D_0 \leftarrow \{j_{\min}, \ldots, j_{\max}\}$.
3. Define $\tilde{c}(j) = \begin{cases} w^{i+1}_{\tau_j} & j \in D_0 \\ w^{i+1}_{\tau_j} + \delta_{j_{\max}, j} & j > j_{\max} \\ \infty & \text{otherwise} \end{cases}$
4. Set $\ell_{i+1} \leftarrow \operatorname{argmin}_{j \geq j_{\min}} \tilde{c}(j)$.

Algorithm 1: "Follow the traversal" algorithm that allows going to all the states between its current state and the current state of the traversal algorithm "for free".

the task; That is, after agent i has selfishly used the system, the mechanism only observes S_i (the state in which the agent used the system) and $w^i_{S_i}$, while the other coordinates of w^i, $(w^i_1, \ldots, w^i_{S_i-1}, w^i_{S_i+1}, \ldots, w^i_m)$, remain unknown.

We resolve this issue by giving the pricing scheme a different (imaginary) sequence of tasks $\tilde{\mathbf{w}}$ with the following *consistency* properties:

1. For every i, $w^i_{S_i}$ is equal to $\tilde{w}^i_{S_i}$.
2. For every i and for every $j \in S \setminus \{S_i\}$, $w^i_j \geq \tilde{w}^i_j$.
3. For every i, given the pricing function P_{i-1} over the state of the system after observing events $1, \ldots, i-1$, the computed vector $\tilde{w}^i$ is consistent with the agent selfishly choosing to process her task in state S_i; I.e., for all $j \neq S_i$,
$$\tilde{w}^i_j + d_{S_{i-1},j} + P_{i-1}(j) \geq w^i_{S_i} + d_{S_{i-1},S_i} + P_{i-1}(S_i). \tag{2}$$

We claim that by running Algorithm 1 on the sequence of tasks $\tilde{\mathbf{w}}$, we get a 2 approximation to the cost of the fractional traversal algorithm on the sequence of tasks $\mathbf{w}$. This clearly holds, since from Lemma 1 we get that the cost of the traversal algorithm given $\tilde{\mathbf{w}}$ cannot be bigger than its cost given $\mathbf{w}$, and from Theorem 3 we get that Algorithm 1 gives a 2 approximation to the cost of the traversal algorithm given sequence $\tilde{\mathbf{w}}$.

Therefore, if we can find a pricing scheme that makes the agents follow Algorithm 1 on sequence $\tilde{\mathbf{w}}$, we ensure that the cost of the incoming agents is the same as that of Algorithm 1 while handling sequence $\tilde{\mathbf{w}}$, hence, we get a 2 approximation to the cost of the traversal algorithm given sequence $\mathbf{w}$.

We show how to compute such $\tilde{\mathbf{w}}$ in the full version. From now on, we assume such $\tilde{w}^i$ can be computed after observing agent i's greedy behavior. We show how to devise a pricing scheme that given $\tilde{\mathbf{w}}$, makes incoming agents act *as though* they are being guided by Algorithm 1 given input $\tilde{\mathbf{w}}$. We describe how to do this in Pricing Scheme 1. We note that although Pricing Scheme 1 may price states using negative number, we can increase all prices by a large enough constant, and not effect the decisions of the agents.

Theorem 4. *Pricing Scheme 1 makes greedy agents simulate Algorithm 1 with task sequence $\tilde{\mathbf{w}}$ given as input.*

Corollary 5. *Pricing Scheme 1 yields a price of anarchy of* $16(m-1)$.

Input: A traversal sequence τ, a set of states S, a distance metric d and an observed sequence of decisions $(S_1, w^1_{S_1}), (S_2, w^2_{S_2}), \ldots$ made by the agents.
Let ℓ_i and t_i be two indices in the traversal sequence, set after the ith agent's decision. Initialize $\ell_0 \leftarrow 1$ and $t_0 \leftarrow 1$.

Setting prices for agent $i+1$:

1. Let $j_{\min} \leftarrow \min(\ell_i, t_i)$ and $j_{\max} \leftarrow \max(\ell_i, t_i)$.
2. Let $D_0 \leftarrow \{j_{\min}, \ldots, j_{\max}\}$.
3. For a state s, let $m(s) \leftarrow \min\{j \geq j_{min} | \tau_j = s\}$.
4. Define $P_i(s) : \begin{cases} -d_{S_i,s} & m(s) \in D_0, \\ -d_{S_i,s} + \delta_{j_{\max}, m(s)} & \text{otherwise.} \end{cases}$

Updating ℓ_{i+1} and t_{i+1} after observing $(S_{i+1}, w^{i+1}_{S_{i+1}})$:

1. Set $\ell_{i+1} \leftarrow m(S_{i+1})$.
2. Compute $\tilde{w}^{i+1}$ as described in the full version.
3. Update t_{i+1} according to the traversal algorithm using $\tilde{w}^{i+1}$ as input.

Pricing Scheme 1: A dynamic pricing scheme that makes incoming agents "follow" Algorithm 1.

4 Pricing Servers on a Line

The k-server problem, introduced by Manasse et al. [12], is the following. Let (V, d) be a fixed metric space. There are k servers, initially located in specified points of V, that can be moved by the algorithm. Moving a server from x to y costs $d(x, y)$. The goal is to serve a sequence of arriving requests with minimum possible cost. Each request is a point in V, and to serve it, the algorithm has to move (at least) one server to that point.

In our setting, we assume that the requests are made by successive agents. Moreover, we assume that each agent is greedy, and will therefore ask that their requests is served with the server that minimizes the sum of its distance to the request and the price we set for the server.

On a high level, we want to find out if any algorithm can be simulated by appropriate pricing, i.e., whether given an algorithm A and its current state (locations of servers), we can give a pricing of servers that will make an arriving greedy agent with a request r, serve r in the same way that A would have, wherever the request occurs.

For some algorithms, designing a payment scheme is straightforward. For instance, Irani and Rubinfeld proved that the following 2-server algorithm *Balance2* is 10-competitive on any metric space [9]. The algorithm keeps track of the total distance that each of the two servers has travelled, and serves each request with the server that minimizes the sum of its distance to the request and *half* the total distance it has travelled before. Clearly, setting the price of each server to the second summand, i.e., half the total distance the serves has moved already, will make the agents follow the algorithm. The only catch is that we have no control over tie-breaking since it is done by the agents. However, this can only affect the total cost by an additive constant.

In general, however, simulating any algorithm A is not as easy. For instance, A could move many servers (in a non-lazy fashion) upon a request. This is the case with some known algorithms, for example with the *Double Coverage* (DC) algorithm [4] that is k-competitive for the line metric. (This algorithm has been generalized to the case of all tree metrics, retaining the optimal ratio [5].) As an agent is only allowed to pick a server that will serve the request, such complex algorithms cannot be simulated directly.

In this work, we restrict our attention to k-server on the tree metric, and the *D*ouble Coverage algorithm in particular. We observe that the standard transformation that makes it a *lazy* algorithm, preserves its other properties, which we call *l*ocality and *m*onotonocity (all defined later), and that combined these enable simulation by a pricing scheme. We remark here that, in general, such transformation employs keeping track of "virtual positions" of servers and basing decisions on those. This makes designing a pricing scheme to simulate the algorithm challenging, since an agent's decision is based on the "real positions" of the servers (and their prices).

4.1 Notation

Given an algorithm A, let $S_A = \{s_1, \ldots, s_k\}$ denote its *state*, i.e., the set of points where A's servers are currently located. We will associate the servers with the points they are in. Similarly, we will associate a request r with the point in which it occurs. Moreover, we associate every point on the real line with a real number. Consequently, we will write $v \leq w$ to denote that v lies to the left of w (or coincides with it) and denote the distance between v and w by $|v-w|$. We use the notation $[v,w]$ to denote the interval spanned by the points v and w, *even if* $v > w$. Given a point v and a finite set of points X, we call the (at most) two points of X, $\max\{x \in X \mid x \leq r\}$ and $\min\{x \in X \mid x \geq r\}$ *adjacent* to r; note that the first one does not exist if $r < \min X$, the second does not exist if $r > \max X$, and they coincide if $r \in X$.

A deterministic online algorithm is *lazy* if the only moves it makes consist in moving a single server to the current request. An algorithm is *local* if it always serves a request with a server that is adjacent to it. Furthermore, an algorithm A is *monotone* if it satisfies the following property: given S_A and a request (at) r, if A would serve a request at r with $s \in S_A$, then it would also serve every request in $[r,s]$ with s.

Later, we give a transformation that takes an algorithm A and returns an "equivalent" algorithm A' that is lazy. This transformation relies on on minimum cost (perfect, bipartite) matching of S_A to $S_{A'}$, where the costs are specified by the metric distance d; in our case, the cost of matching $s \in S_A$ to $s' \in S_{A'}$ is $d(s,s') = |s-s'|$. Given two sets of points X and Y of equal cardinality, we denote any such matching by $M(X,Y)$ and its cost by $d(X,Y)$. In our case, when the metric space is a line, we focus on a particular minimum cost perfect matching (see full version).

4.2 Algorithm transformation

We discuss how every algorithm A can be turned into an equivalent one that is both lazy and local, and never incurs larger cost than A. Later we will show that, for DC, the transformation also preserves monotonicity.

Input: An algorithm A and an online sequence of requests $r_1, \ldots, r_i$.
Output: A lazy algorithm A'.

1. Let S_A be the state of A after serving r_i and $S_{A'}$ the state of A' right before serving r_i; note that $r_i \in S_A$.
2. Find $M = M(S_A, S_{A'})$ and serve r_i with the server matched to it in M.

Algorithm 2: The algorithm transformation.

In general, the transformation only guarantees that its output is a lazy algorithm of no larger cost that A. The additional properties rely on particular structure of the input algorithm and the matching that is used upon requests.

Lemma 6. *Given an algorithm A, the transformation produces a lazy algorithm A' such that for every sequence of requests $R = r_1, \ldots, r_i$, it holds that $C(A',R) \leq C(A,R)$. A' is also local if, upon each request r_i, a matching with certain additional properties is used.*

Note that we have proved that the algorithm that our transformation produces is local but not that it is monotone. Nevertheless, for the case of line and the *Double Coverage* (DC) algorithm [4, 5], we can prove monotonicity.

4.3 Transforming DC on a line

The *Double Coverage* (DC) algorithm has been originally designed for the line metric [4], and later generalized to all tree metrics [5]. For theses metrics, it attains the optimal competitive ratio of k. Its description is given in the full version.

To prove that DC on a line can be transformed to a lazy local monotone algorithm, we will rely on the fact that DC itself is local and monotone, as well as on properties of canonical matchings on a line.

Lemma 7. *Applied to DC on a line, the transformation produces a lazy local monotone algorithm if, upon each request r_i, a particular matching is used.*

4.4 Payment scheme

We define a payment scheme as $P\colon S_A \mapsto \mathbb{R}$. The disutility of an agent while serving a request r using server s is defined as $d(r,s)+P(s)$. We assume that at each iteration the request r is served using the server s that minimizes $d(r,s)+P(s)$, where r is the request the incoming agent wishes to serve.

Theorem 8. *For every lazy local monotone deterministic k-server algorithm A on a line, there exists a payment scheme P that makes incoming agents* weakly *follow A and incur movement cost no larger than that of A.*

See the full version (appended) for explanation of what "weakly" means.

Corollary 9. *There exists a payment scheme for the k-server problem on a line which yields a price of anarchy of k.*

5 Pricing Parking Slots on a City Block

We consider the following setting. The "town" is modeled as an undirected graph $G=(V,E)$, $|V|=m$. Vertices of G represent empty parking slots, which are the possible destinations of drivers arriving in town. Every edge $e=(u,v)\in E$ has an associated weight w_e, corresponding to the distance between vertices v and u. In this model, $n\leq m$ cars arrive online, cars behave selfishly and seek to minimize their disutility (defined below).

A dynamic pricing scheme $P=(P_1,P_2,\ldots,P_n)$ where $P_i:\{x(1),x(2),\ldots,x(i-1)\}\times V\mapsto\mathfrak{R}$ sets a price for an unoccupied parking space $v\in V$ given that there are $i-1$ occupied parking spaces: $x(1),\ldots,x(i-1)$. The functions P_i may use random bits. Set $P_0(v)=0$ for all $v\in V$ (or some other constant). When we refer to $P_i(v)$, we assume that $i-1$ cars have already arrived and are parked at $x(1),\ldots,x(i-1)$. Thus,

- Given G and $i\in 1,\ldots,n$, and given that cars $j\in 1,\ldots,i-1$ are already parked at locations $x(1),\ldots,x(i-1)$, $P_i(v)\in\mathfrak{R}$ gives a price to park at $v\in V\setminus\{x(1),\ldots,x(i-1)\}$. This is a one-time parking fee, not (say) charged by the hour.
- $P_i(v)$ can only be queried after $i-1$ cars have parked, and the responses are consistent as long as no additional cars park. The current parking fees, $P_i(v)$, $v\in V\setminus\{x(1),\ldots,x(i-1)\}$ are known to car i *before* parking.

An assignment $x : [n] \mapsto V$ maps cars $i \in 1, \ldots, n$ to *parking slots*. This can be a online greedy assignment (determined by selfish agents) or an arbitrary assignment.

Incentives for cars are determined as follows:

- Let $g(i)$, $1 \leq i \leq n$ the be destination of car i, g is called the *goal function*.
- Given the graph G, goal function g, and assignment function x, the *non-monetary cost* to agent $i \in [n]$, $C_i(G,g,x)$, is a function of the destination $g(i)$, and the parking slot $x(i)$. $C_i(G,g,x) = d\left(g\left(i\right), x\left(i\right)\right)$, where $d\left(g\left(i\right), x\left(i\right)\right)$ is the weight of a minimum weight path from $x(i)$ to $g(i)$ in G. Note that $C_i(G,g,x)$ does not include any parking fee component.
- The disutility of agent i (who chooses to park at unoccupied slot $x(i)$) is $C_i(G,g,x) + P_i(x(i))$. *I.e.*, the disutility is the sum of the non-monetary cost to agent i plus the parking fee to park at $x(i)$.

Given any pricing scheme P, define X_{eq} to be the set of assignments in equilibria: for $i = 1, \ldots, n$, given P_i, car i chooses $x(i)$ to minimize its disutility, breaking ties arbitrarily.

Social costs are defined as follows:

- Given G and g, the social cost of an assignment x is the sum of the *non-monetary costs* to the drivers: $C(G,g,x) = \sum_{i \in [n]} C_i(G,g,x)$.
- Fix G and a goal function g and let x_{opt} be an assignment that minimizes the social cost. Fix a pricing scheme P and let X_{eq} be the set of assignments in equilibria. Note that X_{eq} depends on the random bits used in P, while the goal function g does not.

$$PoA(G,P) = \sup_g \frac{\mathrm{E}_{\text{Random bits in } P}\left(\sup_{x \in X_{eq}} C(G,g,x)\right)}{C(G,g,x_{\text{opt}})}.$$

5.1 Dynamic Harmonic Pricing for the Weighted Line Graph

In this section we describe the *harmonic pricing scheme* that guaranties an $O\left(\min\left\{\log R, n^2\right\}\right)$ approximation to the minimal cost matching *in expectation*, where R is the aspect ratio of the underlying metric. The expectation is over the random coins of the mechanism. The coordination mechanisms sets fees to the vacant parking slots. The payments are devised without knowing the goals of the drivers. Once set, a car parks so as to minimize its disutility.

The improved approximation guaranties of this mechanism should be compared with the $\Omega(2^n)$ approximation lower bound for the greedy matching on the weighted line graph with free parking [10]. On the same input, the *harmonic pricing scheme* incurs a cost of n in expectation.

The pricing is probabilistic and guaranties that the best response of next car to arrive will be one of the following: If there is a free spot at her goal, she will park there. Otherwise, she will either park at the closest unoccupied parking spot to the right of her goal, or, park at the closest unoccupied parking spot to the left of her goal.

5.1.1 The Harmonic Algorithm

In the online matching settings, there is a metric space (V,d) with k predefined servers $S \subseteq V$. A set $R = r_1, r_2, \ldots, r_k$ of requests arrive online one after another, and for each incoming request the algorithm needs to choose a previously unmatched server to handle the request. The goal is to minimize the sum of distances the servers need to move in order to handle the incoming request. Gupta and Lewi [7] gave a natural algorithm for this problem on a line metric: let d_l be the distance of the incoming request from the nearest unmatched server to the left of it, and d_r be the distance from the nearest unmatched server to the right. The algorithm assigns the request to the server on the left with probability $\frac{d_r}{d_l + d_r}$, and to the server on the right with the remaining probability. They gave a proof of the following guaranty about the performance of the harmonic algorithm:

Theorem 10 ([7]). *Let $d_{\max}$ be the maximal distance between two vertices, and let $d_{\min}$ be the minimal one. The competitive ratio of the harmonic algorithm on a line metric is $O\left(\log \frac{d_{\max}}{d_{\min}}\right)$.*

Theorem 10 yields an $O(\log n)$ approximation algorithm to the min cost matching in the case where quantity $d_{\max}/d_{\min}$, called the *aspect ratio* of the metric, can be bounded by a polynomial in n.

It is easy to see the close connection between the online problem and our game-theoretic problem. Next we show how to devise payments that gives us the same desired behavior by the agents, and we see that the same approximation guaranty still holds.

5.1.2 Computing the posted parking fees

We now show how to set prices to the vacant parking slots that effectively makes incoming [selfish] agents park *as though* they were obeying orders given by the online harmonic algorithm.

We say that a pricing scheme *ensures harmonic behavior* if, when an agent arrives:

- She chooses to park at her goal if vacant.
- Otherwise, she chooses between the two closest vacant slots to the left and right of her goal with harmonic probabilities.

A maximal contiguous run of occupied parking spaces is called a *block*.

Just before car i is set to park let $V' = \{v_1, \ldots, v_k\}$ the set of vacant parking spots and let $B_1, B_2, \ldots, B_\ell \subseteq V$ the current set of "blocks". We use $L(B_j)$ to denote the first available vertex to the left of block B_j, for some $j \in \{1, \ldots, \ell\}$, and let $R(B_j)$ denote the first available vertex to the right of B_j. Let $d_j = d(L(B_j), R(B_j))$, we define a set of distributions $D_1, D_2, \ldots, D_\ell$, where for every $j \in \{1, \ldots, \ell\}$, D_j is uniformly distributed on the real interval $[-d_j, d_j]$.

Let $P : V' \mapsto \mathbb{R}^+$ be a random function that sets prices to vacant slots. Previously, this function was denoted P_i, but we omit the index i for the remainder of this section.

We say that P satisfies the *Harmonic Payment Conditions* if:

1. For any $j \in \{1, \ldots, \ell\}$, $P(L(B_j)) - P(R(B_j)) \sim D_j$.
2. For any $u, v \in V'$ s.t. there are no unoccupied parking slots between them: $|P(u) - P(v)| < d(u, v)$.

Lemma 11. *Any pricing function P that satisfies the* Harmonic Payment Conditions *also ensures harmonic behaviour on the part of arriving cars, regardless of their goals.*

Next, we show a specific pricing scheme that satisfies the harmonic payment conditions.

Observation 12. *Let $R(v)$ be the set $\{j \in \{1, \ldots, \ell\} : v \text{ is left of } B_j\}$ and let $q_1 \sim D_1, g_2 \sim D_2, \ldots, q_\ell \sim D_\ell$. For any constant c, setting $P(v) = c + \sum_{j \in R(v)} q_j$ satisfies the* Harmonic Payment Conditions*.*

We can use the constant c in order to obtain a pricing scheme that satisfies the harmonic payment conditions in which there are no negative transfers. In the full version, we show how to compute prices which satisfy desirable objectives (e.g. minimizes the sum of prices of all vacant parking slots) using an LP. For general weighted line graphs, without further assumptions, we get the following:

Theorem 13. *Any pricing scheme which satisfies the harmonic payment conditions gives an $O\left(\min\{\log \frac{d_{\max}}{d_{\max}}, n^2\}\right)$ approximation to the optimal assignment in weighted line graphs in expectation.*

In the full version (appended), we show how to devise a pricing scheme that guarantees an $O(\log n)$ approximation to optimal social cost given its poly(n) factor estimate.

References

[1] Baruch Awerbuch, Yossi Azar, and Adam Meyerson. Reducing truth-telling online mechanisms to online optimization. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 503–510, New York, NY, USA, 2003. ACM.

[2] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Online auctions and generalized secretary problems. *SIGecom Exchanges*, 7(2), 2008.

[3] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[4] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[5] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[6] Michael Cooper. A meter so expensive, it creates parking spots. `http://nyti.ms/1dIvpA0`, 2012.

[7] Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *ICALP*, pages 424–435, 2012.

[8] Mohammad Taghi Hajiaghayi, Robert Kleinberg, and David C. Parkes. Adaptive limited-supply online auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, pages 71–80, New York, NY, USA, 2004. ACM.

[9] Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Inf. Process. Lett.*, 39(2):85–91, 1991.

[10] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.

[11] Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 233–241, New York, NY, USA, 2000. ACM.

[12] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[13] David C. Parkes. Online mechanisms. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16, pages 411–439. Cambridge University Press, New York, NY, USA, 2007.

[14] D.C. Shoup and American Planning Association. *The High Cost of Free Parking*. Planners Press, American Planning Association, 2005.